```
  On aborde la traduction des constructeurs algorithmiques.
  On privilégie la traduction mécanique (travail du compilateur).

I) Branchements

I.1) Définition

  Un saut ou branchement est une affectation au compteur ordinal (PC).
  Effet : passage à une autre instruction que celle qui suit en séquence.

I.2) Relatif versus absolu

  Sauter n instructions :
     Pc <- PC + n*4   @ Saut relatif        b ou bal (branch always)

  Aller à l'instruction à l'adresse A :
     PC <- A          @ Saut absolu        autres processeurs jmp (jump)
                                           ARM : PC est accessible comme R15
                                           ldr  R15, =adresse

I.3) Effet du pipeline

  Travail à la chaîne : écrire resultat i-2 // calculer i-1 // lire i

  Lecture de PC comme r15 : pointe 2 instructions en avance (pour toute
utilisation de r15 comme opérande des instructions de calcul et dans les
[] de ldr et str). La valeur de PC lue est adresse de instruction courante + 8.

I.4) Sens de branchement et boucles

  Adresse de l'intruction >  Adresse de l'instruction de branchement : avant
                          <=                                       : arrière

  Branchement relatif en arrière : n <= 0


  Les branchements en arrière créent des boucles : le processeur réexécute
les instructions qui précèdent. Si branchement arrière inconditionnel : boucle
infinie.

Exemple :

Adresse    Etiquette    Instruction            Signification

10000    debut:       mov    r1, #0            @ r1 <- 0
10004    dest_rel:    sub    r1,r1,#1          @ r1 --
10008    saut_abs:    ldr    r15, =dest_abs    @ pc <- dest_abs
1000C                 mov    r1, #4            @ r1 <- 4 (jamais executée)
10010    dest_abs:    add    r1,r1,r1          @ r1 = r1 * 2
10014    saut_rel:    b      dest_rel          @ pc <-  -6 instructions
10018                 ^
1001C                 v  + 2 instructions      @ depuis ici (PC + 8)
                      .ltorg

Ordre d'execution (derniers chiffres d'adresses) :

00 04 08 10 14 04 08 10 14 04 08 10 ...

Cas du ARM : déplacement de n instructions, n codé sur 24 bits : + ou - 8 Mi ins
tructions
```



```
I.6) Branchements conditionnels

  Si condition = expr (Z,N,C,V) vraie alors PC <- PC + déplacement*4
                                sinon       PC <- PC + 4
II) Si alors sinon

On introduit les goto et les etiquettes pour faire apparaitre les branchements.
On fait apparaitre la comparaison + on INVERSE la condition :
si condition fausse sauter au sinon.

Exemple

register int x,y;        /* Entiers signes */

             x == y  x != y           not (x-y)==0
if (x==y)          |     |    compar:    if ((x - y) != 0) goto sinon
  {                |     .
  x = x + 2;       |     .    alors:    x = x + 2;
  y = y - 1;       |     .              y = y - 1;
  }                |     .              goto fin_si;
else               .     v
  {                .     |
  y = y - 3;       .     |    sinon:    y = y - 3;
  x = x + 5;       .     |              x = x + 5;
  }                v     |
x = x + y;         |     |    fin_si:   x = x + y;

Traduction ARM :  x : r1  y : r2

compar:       cmp    r1, r2          @ ZNCV = f(x-y)
              bne    sinon           @ brancher si Z=0 (resultat != 0)
alors:        add    r1, r1, #2
              sub    r2, r2, #1
              b      fin_si          @ supprimer pour si alors
sinon:        sub    r2, r2, #3      @ supprimer pour si alors
              add    r1, r1, #5      @ supprimer pour si alors
fin_si:       add    r1, r1, r2

L'étiquette compar n'est pas nécessaire (lisibilité uniquement).

III) Si alors et remarques diverses

  Dans les test <,<=,>,>=, tenir compte de la nature des entiers comparés.
Exemple : if (x<=y) goto etiquette :

+ adresses ou entiers naturels : condition sur Z et C          (bls etiquette)
+ entiers relatifs             : condition sur Z et N ouex V   (ble étiquette)

  Le programmeur utilise une étiquette, l'assembleur calcule le déplacement
entre l'instruction de branchement et l'étiquette destination du saut.

  Ce branchement ne sert à rien (erreur de traduction ou simplification) :
       if (x < y) goto xinfy            cmp  r0,r1     @ x rdans r0, y dans r1
                                        blo  xinfy     @ saut totalement inutile
xinfy: y = y-x;                 xinfy:  sub  r1,r1,r0

  Condition x<y vraie   : sauter à sub r1,r1,r0 à l'étiquette xinfy
  Condition x<y gfausse : passer à l'instruction suivante : sub r1,r1,r0 aussi
```

```
  Traduction du if sans else :
                cas particulier de
if (x==y) {     if (x==y) {             if ((x-y)!=0) goto sinon
   x= x+2;         x=x+2;                   x=x+2;
   y = y-1;        y=y-1;                   y=y-1;
}               } else {                    goto finsi
                /* sinon vide */    sinon:
                }
x = x+y;        x=x+y;              finsi:  x=x+y;

Etiquettes sinon et finsi synonymes :   if ((x-y) != 0) goto finsi
                                           x=x+2;
                                           y=y-1;
                                           goto finsi
                                    finsi:  x=x+y;


Le goto finsi ne sert à rien

if (x==y) {         if ((x-y) != 0) goto finsi              cmp r0,r1
                                                            bne finsi
  x=x+2;            x=x=2;                                  add r0,r0,#2
  y=y-1;            y=y-1;                                  sub r1,r1,#1
}
x=x+y;     finsi:   x=x+y;                           finsi: add r0,r0,r1


IV) Répéter tant que (do ... while)

Après chaque exécution du corps, on teste la condition et on rebranche
au début du corps. Le corps est toujours exécuté au moins une fois.

do                                                          |
  {                                                         |
  x = x + 2;       corps:    x = x + 2;                     |<..
  y = y + 1;                 y = y + 1;                     |  | cond vraie
} while (x < y);   compar:   if ((x-y) <0) goto corps;      |...
x = x + y;         fin_do:   x = x + y;                     |

V) Tant que

Idem do, mais ajouter un saut au test pour vérifier la condition avant
la première exécution du corps.

Avec  test de la condition non inversée après le corps de boucle :
un saut conditionnel (if .. goto) par tour de boucle

while (x < y) {                 goto condwhile
  x = x+2;          corpswhile : x = x+2;
  y = y+1;                       y = y+1;
}                   condwhile:   if (x-y) < 0) goto corpswhile
x = x+y;                         x+x+y;


Variante avec test avant le corps  de boucle et condition INVERSEE
                                                        ********

A chaque tour de boucle : un branchement condtionnel INVERSE :
   if (!cond) goto fin
et un branchement inconditionnel
   goto corps) après le corps.
```



```
                    test:    if condition_inversée goto fin
                             corps
                             goto test;
                    fin:    @ suite

while (x<y) {       condw:   if (x -y) >= 0) goto finw
  x=x+2;                     x = x+2;
  y=y+1;                     y = y+1;
}                            goto condw
x=x+y:               finw:   x = x+y

Version avec test avant semble plus naturelle, mais plus "piégeuse" :

1) Oublier d'inverser ou mal inverser la condition dans le test
    --> la condition inverse de x<y n'st pas x>y mais x >= y

2) Oublier le branchement conditionnnel sur le test après le corps.

VI) Parcourant/for

Syntaxe C :    for (initialisation;cond_continuation;mise_a_jour)
           {
           corps_du_for;
           }

Exemple :  facto := 1;
           Pour indice parcourant 1 .. n
               facto = facto * indice;

           for (indice = 1; indice <= n; indice ++) {
           facto = facto * indice;
           }

==> transformation en while équivalent :

        indice = 1;             /* initialisation */
        while (indice <= n) {
           facto *= indice;     /* corps du for */
           indice ++;           /* mise a jour  */
           }

VII) Conditions composées : && (etpuis), || (oubien)

/* Ne pas traiter si manque de temps */
Exemple :

if((a == b) && (c>d)) {       if (a-b) != 0) goto sinon;
                              if (c<=d) goto sinon;
  x = y;            alors:    x = y;
  } else {                    goto fin_si;
  y = x;            sinon:    y = x;
  }
x++;                fin_si:    x++;

if((a == b) || (c>d)) {       if (a-b) == 0) goto alors;
                              if (c<=d) goto sinon;
  x = y;            alors:    x = y;
  } else {                    goto fin_si;
  y = x;            sinon:    y = x;
  }
x++;                fin_si:    x++;
```